

```
import pandas as pd
import seaborn as sns
import os
import numpy as np
import matplotlib.pyplot as plt
```

```
housing_df = pd.read_csv('/content/housing.csv')
```

```
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   longitude              20640 non-null  float64
1   latitude               20640 non-null  float64
2   housing_median_age     20640 non-null  float64
3   total_rooms            20640 non-null  float64
4   total_bedrooms         20433 non-null  float64
5   population             20640 non-null  float64
6   households              20640 non-null  float64
7   median_income          20640 non-null  float64
8   median_house_value     20640 non-null  float64
9   ocean_proximity        20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
housing_df.shape
```

```
(20640, 10)
```

```
housing_df.head()
```


	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	

Next steps:

[Generate code with housing_df](#)

 [View recommended plots](#)

```
housing_df.tail()
```



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	

```
housing_df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	142
std	2.003532	2.135952	12.585558	2181.615252	421.385070	113
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	78
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	116
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	172
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	3568

```
housing_df.isnull().sum()
```

```
longitude      0
latitude       0
housing_median_age  0
total_rooms    0
total_bedrooms 207
population     0
households     0
median_income  0
median_house_value  0
ocean_proximity  0
dtype: int64
```

```
housing_df['total_bedrooms'].isnull().sum()/housing_df.shape[0] * 100
```

```
1.002906976744186
```

```
from sklearn.impute import KNNImputer
```

```
# create a temporary copy of the dataset
```

```
housing_df_temp = housing_df.copy()
```

```
# retrieve columns with numerical data; will exclude the ocean_proximity column since the datatype is object; other columns are float64
columns_list = [col for col in housing_df_temp.columns if housing_df_temp[col].dtype != 'object']
```

```
# extract columns that contain at least one missing value
```

```
new_column_list = [col for col in housing_df_temp.loc[:, housing_df_temp.isnull().any()]]
```

```
# update temp dataframe with numeric columns that have empty values
```

```
housing_df_temp = housing_df_temp[new_column_list]
```

```
# initialize KNNImputer to impute missing data using machine learning
```

```
knn = KNNImputer(n_neighbors = 3)
```

```
# fit function trains the model
```

```
knn.fit(housing_df_temp)
```

```
# transform the data using the model
```

```
# applies the transformation model (ie knn) to data
```

```
array_values = knn.transform(housing_df_temp)
```

```
# convert the array values to a dataframe with the appropriate column names
```

```
housing_df_temp = pd.DataFrame(array_values, columns = new_column_list)
```

```
housing_df_temp.isnull().sum()
```

```
total_bedrooms    0
dtype: int64
```

```
for column_name in new_column_list:
```

```
    housing_df[column_name] = housing_df_temp.replace(housing_df[column_name],housing_df[column_name])
```

```
# confirm columns no longer contain null data
```

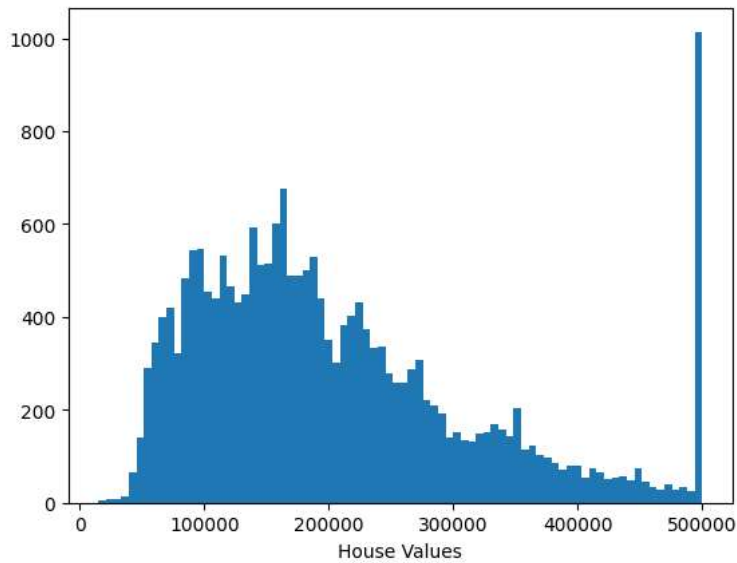
```
housing_df.isnull().sum()
```

```
longitude      0
latitude       0
```

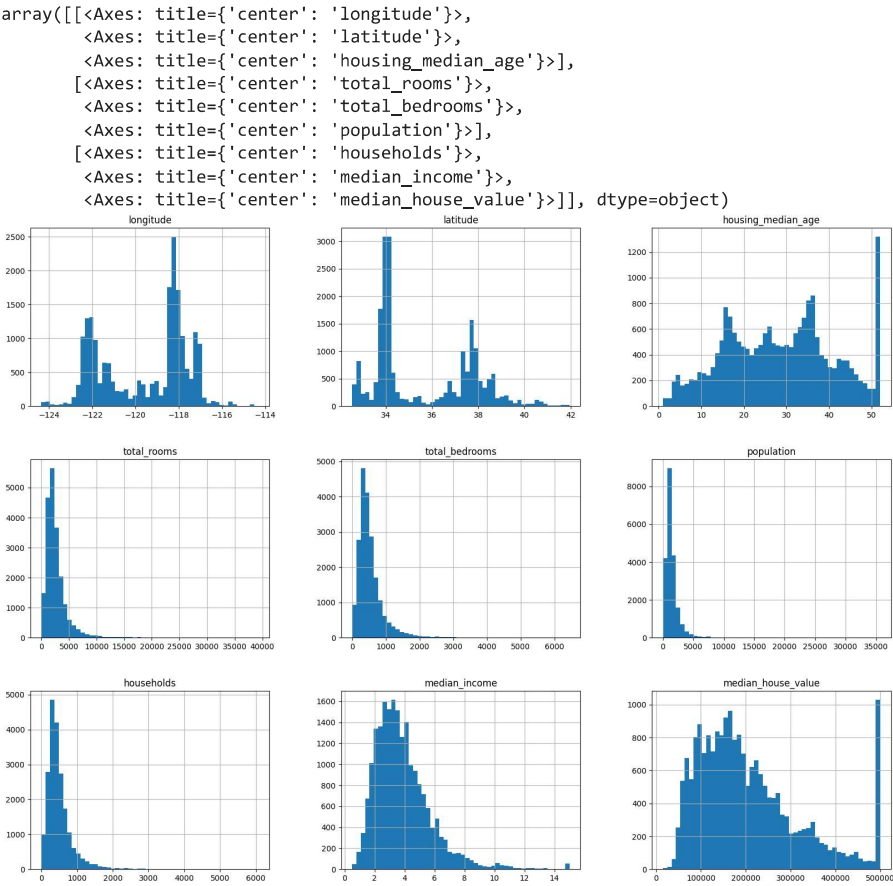
```
housing_median_age    0
total_rooms            0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

```
plt.hist(housing_df['median_house_value'], bins=80)
plt.xlabel("House Values")
```

Text(0.5, 0, 'House Values')



```
housing_df.hist(bins=50, figsize=(20,15))
```



```
corr = housing_df.corr() # data frame correlation function
print(corr)
```

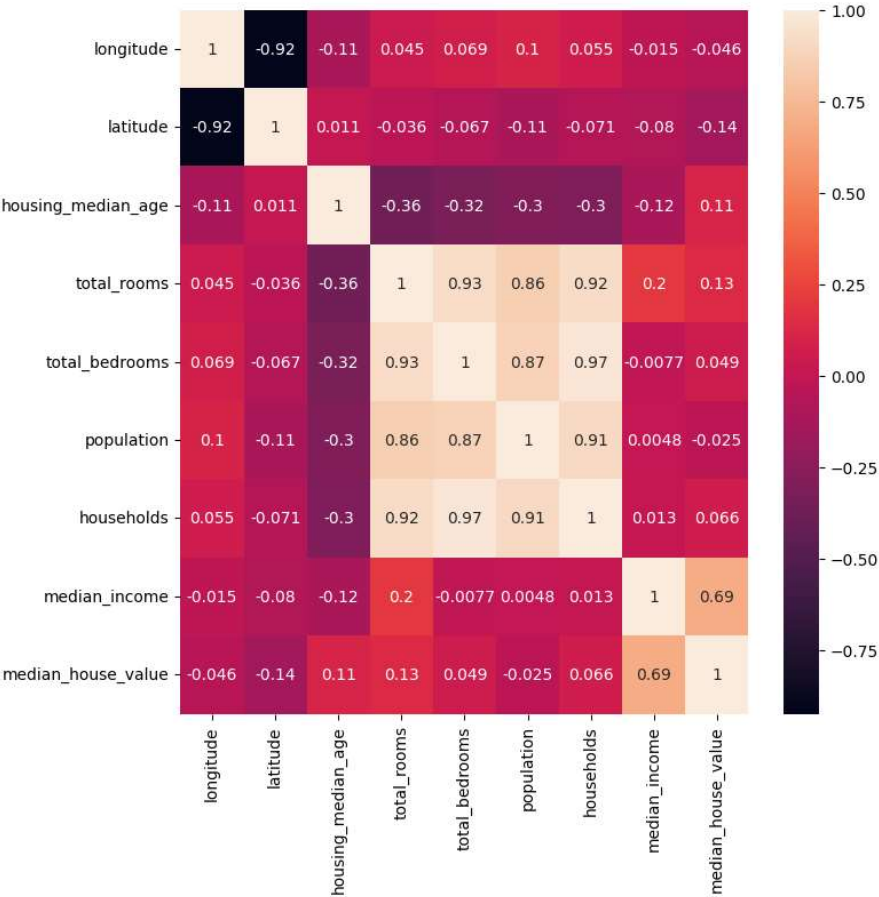
	longitude	latitude	housing_median_age	total_rooms	\
longitude	1.000000	-0.924664	-0.108197	0.044568	
latitude	-0.924664	1.000000	0.011173	-0.036100	
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	
total_rooms	0.044568	-0.036100	-0.361262	1.000000	
total_bedrooms	0.069260	-0.066658	-0.318998	0.927253	
population	0.099773	-0.108785	-0.296244	0.857126	
households	0.055310	-0.071035	-0.302916	0.918484	
median_income	-0.015176	-0.079809	-0.119034	0.198050	

median_house_value	-0.045967	-0.144160	0.105623	0.134153
	total_bedrooms	population	households	median_income \
longitude	0.069260	0.099773	0.055310	-0.015176
latitude	-0.066658	-0.108785	-0.071035	-0.079809
housing_median_age	-0.318998	-0.296244	-0.302916	-0.119034
total_rooms	0.927253	0.857126	0.918484	0.198050
total_bedrooms	1.000000	0.873910	0.974725	-0.007682
population	0.873910	1.000000	0.907222	0.004834
households	0.974725	0.907222	1.000000	0.013033
median_income	-0.007682	0.004834	0.013033	1.000000
median_house_value	0.049454	-0.024650	0.065843	0.688075

	median_house_value
longitude	-0.045967
latitude	-0.144160
housing_median_age	0.105623
total_rooms	0.134153
total_bedrooms	0.049454
population	-0.024650
households	0.065843
median_income	0.688075
median_house_value	1.000000

<ipython-input-16-68dfa24ced17>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
corr = housing_df.corr() # data frame correlation function

```
plt.figure(figsize = (8,8))  
  
sns.heatmap(corr, annot=True)  
plt.show()
```



```
housing_df['rooms_per_household'] = housing_df['total_rooms']/housing_df['households']

# a new feature that is a ratio of the total bedrooms to the total rooms
housing_df['bedrooms_per_room'] = housing_df['total_bedrooms']/housing_df['total_rooms']

# a new feature that is a ratio of the population to the households
housing_df['population_per_household'] = housing_df['population']/housing_df['households']

# let's combine the latitude and longitude into 1
housing_df['coords'] = housing_df['longitude']/housing_df['latitude']

housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   longitude              20640 non-null  float64
1   latitude               20640 non-null  float64
2   housing_median_age     20640 non-null  float64
3   total_rooms            20640 non-null  float64
4   total_bedrooms         20640 non-null  float64
5   population             20640 non-null  float64
6   households             20640 non-null  float64
7   median_income          20640 non-null  float64
8   median_house_value     20640 non-null  float64
9   ocean_proximity        20640 non-null  object
10  rooms_per_household    20640 non-null  float64
11  bedrooms_per_room      20640 non-null  float64
12  population_per_household 20640 non-null  float64
13  coords                 20640 non-null  float64
dtypes: float64(13), object(1)
memory usage: 2.2+ MB
```

```
housing_df = housing_df.drop('total_rooms', axis=1)
housing_df = housing_df.drop('households', axis=1)
housing_df = housing_df.drop('total_bedrooms', axis=1)
housing_df = housing_df.drop('population', axis=1)
housing_df = housing_df.drop('longitude', axis=1)
housing_df = housing_df.drop('latitude', axis=1)
```

```
housing_df.info()
```

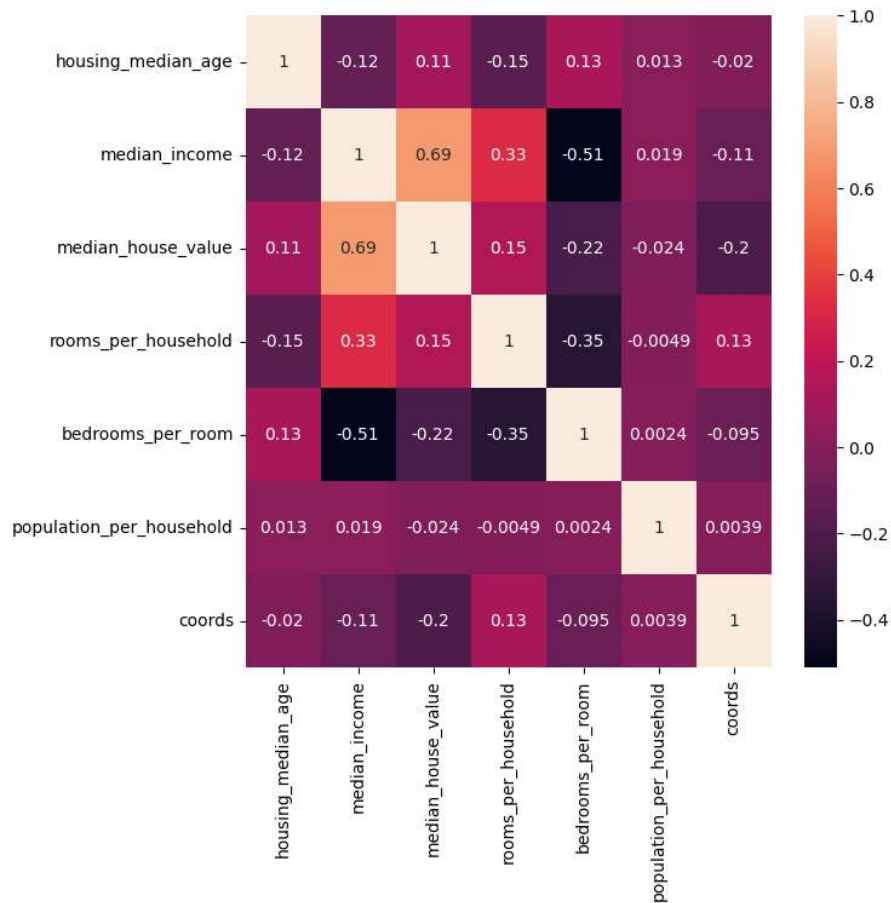
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   housing_median_age     20640 non-null  float64
1   median_income          20640 non-null  float64
2   median_house_value     20640 non-null  float64
3   ocean_proximity        20640 non-null  object
4   rooms_per_household    20640 non-null  float64
5   bedrooms_per_room      20640 non-null  float64
6   population_per_household 20640 non-null  float64
7   coords                 20640 non-null  float64
dtypes: float64(7), object(1)
memory usage: 1.3+ MB
```

```
corr = housing_df.corr()
```

```
#make the heatmap larger in size
plt.figure(figsize = (7,7))
```

```
sns.heatmap(corr, annot=True)
plt.show()
```

```
<ipython-input-20-08fcc2884738>:1: FutureWarning: The default value of numeric_only in l
corr = housing_df.corr()
```



```
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   housing_median_age     20640 non-null  float64
1   median_income          20640 non-null  float64
2   median_house_value     20640 non-null  float64
3   ocean_proximity        20640 non-null  object
4   rooms_per_household    20640 non-null  float64
5   bedrooms_per_room      20640 non-null  float64
6   population_per_household 20640 non-null  float64
7   coords                 20640 non-null  float64
dtypes: float64(7), object(1)
memory usage: 1.3+ MB
```

```
housing_df.ocean_proximity.unique()
```

```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

```
# let's count
```

```
housing_df["ocean_proximity"].value_counts()
```

```
<1H OCEAN    9136
INLAND       6551
NEAR OCEAN   2658
NEAR BAY     2290
ISLAND        5
Name: ocean_proximity, dtype: int64
```

```
print(pd.get_dummies(housing_df['ocean_proximity']))
```

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	1	0
...
20635	0	1	0	0	0
20636	0	1	0	0	0
20637	0	1	0	0	0
20638	0	1	0	0	0
20639	0	1	0	0	0

[20640 rows x 5 columns]

```
housing_df_encoded = pd.get_dummies(data=housing_df, columns=['ocean_proximity'])
```

print the first few observations; notice the old OCEAN_PROXIMITY column is gone

```
housing_df_encoded.head()
```

	housing_median_age	median_income	median_house_value	rooms_per_household	bedrooms_
0	41.0	8.3252	452600.0	6.984127	
1	21.0	8.3014	358500.0	6.238137	
2	52.0	7.2574	352100.0	8.288136	
3	52.0	5.6431	341300.0	5.817352	
4	52.0	3.8462	342200.0	6.281853	

Next steps:

[Generate code with housing_df_encoded](#)

 [View recommended plots](#)

```
import sklearn
from sklearn.model_selection import train_test_split

# remove spaces from column names and convert all to lowercase and remove special characters as it could cause issues in the future
housing_df_encoded.columns = [c.lower().replace(' ', '_').replace('<', '_') for c in housing_df_encoded.columns]

# Split target variable and feature variables
X = housing_df_encoded[['housing_median_age', 'median_income', 'bedrooms_per_room', 'population_per_household', 'coords', 'ocean_proximity__1h_c
    'ocean_proximity_inland', 'ocean_proximity_island', 'ocean_proximity_near_bay', 'ocean_proximity_near_ocean']]
y = housing_df_encoded['median_house_value']

print(X)
```

	housing_median_age	median_income	bedrooms_per_room	\
0	41.0	8.3252	0.146591	
1	21.0	8.3014	0.155797	
2	52.0	7.2574	0.129516	
3	52.0	5.6431	0.184458	
4	52.0	3.8462	0.172096	
...	
20635	25.0	1.5603	0.224625	
20636	18.0	2.5568	0.215208	
20637	17.0	1.7000	0.215173	
20638	18.0	1.8672	0.219892	
20639	16.0	2.3886	0.221185	

	population_per_household	coords	ocean_proximity__1h_ocean	\
0	2.555556	-3.226769	0	
1	2.109842	-3.228209	0	
2	2.802260	-3.229590	0	
3	2.547945	-3.229855	0	
4	2.181467	-3.229855	0	
...	
20635	2.560606	-3.067123	0	
20636	3.122807	-3.069385	0	
20637	2.325635	-3.074309	0	
20638	2.123209	-3.076845	0	
20639	2.616981	-3.079502	0	

	ocean_proximity_inland	ocean_proximity_island	\
0	0	0	


```

1          0          0
2          0          0
3          0          0
4          0          0
...      ...      ...
20635      1          0
20636      1          0
20637      1          0
20638      1          0
20639      1          0

```

```

      ocean_proximity_near_bay  ocean_proximity_near_ocean
0          1          0
1          1          0
2          1          0
3          1          0
4          1          0
...      ...      ...
20635      0          0
20636      0          0
20637      0          0
20638      0          0
20639      0          0

```

```
[20640 rows x 10 columns]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, shuffle=True, test_size=0.3)
```

```
# Confirm how the data was split
```

```
print(X_train.shape)
```

```
print(X_test.shape)
```

```
print(y_train.shape)
```

```
print(y_test.shape)
```

```
(14448, 10)
```

```
(6192, 10)
```

```
(14448,)
```

```
(6192,)
```

```
from sklearn.linear_model import LinearRegression
```

```
# Create a Linear regressor using all the feature variables
```

```
reg_model = LinearRegression()
```

```
# Train the model using the training sets
```

```
reg_model.fit(X_train, y_train)
```

```
▸ LinearRegression
```

 **Generate**

create a dataframe with 2 columns and 10 rows



 

Close

```
y_pred_test = reg_model.predict(X_test)
```

```
pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_test})
```

```
pred_test_df
```

	Actual	Predicted	
20046	47700.0	103743.050896	
3024	45800.0	92451.250932	
15663	500001.0	219490.963844	
20484	218600.0	283292.425471	
9814	278000.0	244228.861575	
...	
17505	237500.0	210121.340663	
13512	67300.0	74907.098235	
10842	218400.0	216609.962950	
16559	119400.0	127975.072923	
5786	209800.0	202803.254310	

6192 rows × 2 columns

Next steps: [Generate code with pred_test_df](#) [View recommended plots](#)

```
r2_reg_model_test = round(reg_model.score(X_test, y_test),2)

print("R^2 Test: {}".format(r2_reg_model_test))

R^2 Test: 0.56

# try another machine learning algorithm : Random Forest
# Use scikit-learn's Random Forest to train the model on both the training and evaluate it on the test sets
from sklearn.ensemble import RandomForestRegressor

# Create a regressor using all the feature variables
rf_model = RandomForestRegressor(n_estimators=10,random_state=10)




# Train the model using the training sets
rf_model.fit(X_train, y_train)
```

▸ RandomForestRegressor

```
y_rf_pred_test = rf_model.predict(X_test)

rf_pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_rf_pred_test})

rf_pred_test_df
```

	Actual	Predicted	
20046	47700.0	47840.0	
3024	45800.0	92680.0	
15663	500001.0	446000.5	
20484	218600.0	265320.0	
9814	278000.0	240800.0	
...	
17505	237500.0	231680.1	
13512	67300.0	69680.0	
10842	218400.0	203930.0	
16559	119400.0	126170.0	
5786	209800.0	198160.0	

6192 rows × 2 columns

Next steps: [Generate code with rf_pred_test_df](#) [View recommended plots](#)

```

from sklearn.metrics import r2_score, mean_squared_error

score = r2_score(y_test, y_rf_pred_test)

print("R^2 - {}".format(round(score, 2) * 100))

R^2 - 75.0%

print('RMSE on test data: ', mean_squared_error(y_test, y_rf_pred_test)**(0.5))

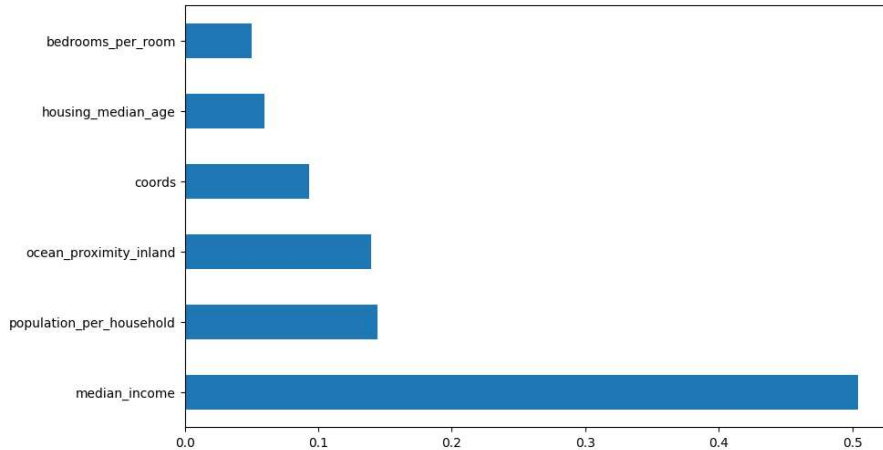
RMSE on test data: 57289.11495447338

```

```

plt.figure(figsize=(10,6))
feat_importances = pd.Series(rf_model.feature_importances_, index = X_train.columns)
feat_importances.nlargest(6).plot(kind='barh');

```



```

train_x_if = X_train[['bedrooms_per_room', 'housing_median_age', 'coords', 'ocean_proximity_inland', 'population_per_household', 'median_income']]
test_x_if = X_test[['bedrooms_per_room', 'housing_median_age', 'coords', 'ocean_proximity_inland', 'population_per_household', 'median_income']]

# create an object of the RandomForestRegressor Model
rf_model_if = RandomForestRegressor(n_estimators=10, random_state=10)

# fit the model with the training data
rf_model_if.fit(train_x_if, y_train)

# predict the target on the test data
predict_test_with_if = rf_model_if.predict(test_x_if)

print('RMSE on test data: ', mean_squared_error(y_test, predict_test_with_if)**(0.5))

RMSE on test data: 57366.910692045196

```

```

pip install xgboost

```

```

Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)

```

```

from xgboost import XGBRegressor

xgb_model = XGBRegressor()

xgb_model.fit(X_train, y_train)

```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
y_xgb_pred_test = xgb_model.predict(X_test)

xgb_pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_xgb_pred_test})


xgb_pred_test_df
```

	Actual	Predicted
20046	47700.0	66404.914062
3024	45800.0	86681.765625
15663	500001.0	449666.093750
20484	218600.0	262887.281250
9814	278000.0	218322.796875
...
17505	237500.0	227466.500000
13512	67300.0	64712.433594
10842	218400.0	218226.109375
16559	119400.0	123181.968750
5786	209800.0	227016.828125

6192 rows × 2 columns

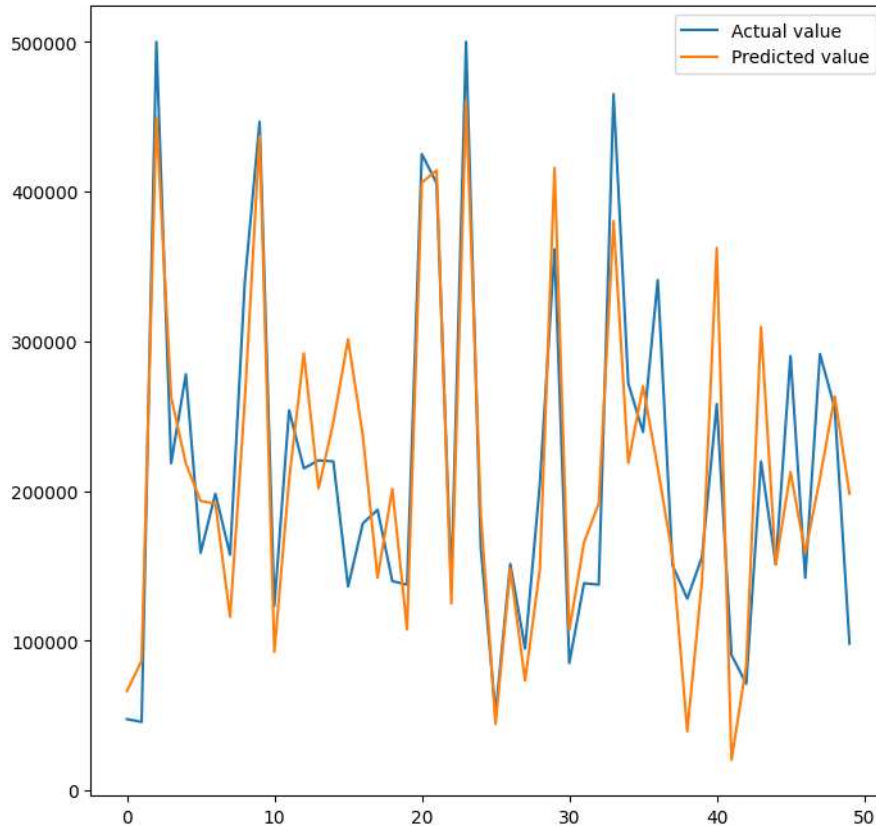
Next steps:

[Generate code with xgb_pred_test_df](#)

 [View recommended plots](#)

```
fig= plt.figure(figsize=(8,8))
xgb_pred_test_df = xgb_pred_test_df.reset_index()
xgb_pred_test_df = xgb_pred_test_df.drop(['index'],axis=1)
plt.plot(xgb_pred_test_df[:50])
plt.legend(['Actual value', 'Predicted value'])
```

<matplotlib.legend.Legend at 0x7a6c06a235b0>



```
from sklearn.metrics import r2_score

score = r2_score(y_test, y_xgb_pred_test)

print("R^2 - {}".format(round(score, 2) * 100))

R^2 - 78.0%
```

```
from sklearn.metrics import mean_squared_error
import math

mse = mean_squared_error(y_test, y_xgb_pred_test)
rmse = math.sqrt(mean_squared_error(y_test, y_xgb_pred_test))

print(mse)
print(rmse)

2939759040.9080276
54219.5448238735
```

```
from sklearn.metrics import mean_absolute_error

print(mean_absolute_error(y_test, y_xgb_pred_test))

36285.050324826894
```

```
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score

# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

scores = cross_val_score(xgb_model, X, y, scoring='r2', error_score='raise', cv=cv, n_jobs=-1, verbose=1)

#average of all the r2 scores across runs
print(scores.mean())
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
0.7850403811484551
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.3s finished
```

```
xgb_model.get_params()
```

```
{'objective': 'reg:squarederror',
 'base_score': None,
 'booster': None,
 'callbacks': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'device': None,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'feature_types': None,
 'gamma': None,
 'grow_policy': None,
 'importance_type': None,
 'interaction_constraints': None,
 'learning_rate': None,
 'max_bin': None,
 'max_cat_threshold': None,
 'max_cat_to_onehot': None,
 'max_delta_step': None,
 'max_depth': None,
 'max_leaves': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'multi_strategy': None,
 'n_estimators': None,
 'n_jobs': None,
 'num_parallel_tree': None,
 'random_state': None,
 'reg_alpha': None,
 'reg_lambda': None,
 'sampling_method': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'validate_parameters': None,
 'verbosity': None}
```

```
xgb_model_2 = XGBRegressor(
    gamma=0.05,
    learning_rate=0.01,
    max_depth=6,
    n_estimators=1000,
    n_jobs=16,
    objective='reg:squarederror',
    subsample=0.8,
    scale_pos_weight=0,
    reg_alpha=0,
    reg_lambda=1,
    verbosity=1)
```

```
xgb_model_2.fit(X_train, y_train)
```

```
#run the predictions on the training and testing data
v xgb 2 pred test = xgb model 2.predict(X test)
```